HW1 Programming Problem 5 (30 points)

Problem Description

Here, you will perform *weighted* KNN regression.

After you write your own code for weighted KNN regression, you will also try out sklearn's built-in KNN regressor.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

Summary of deliverables:

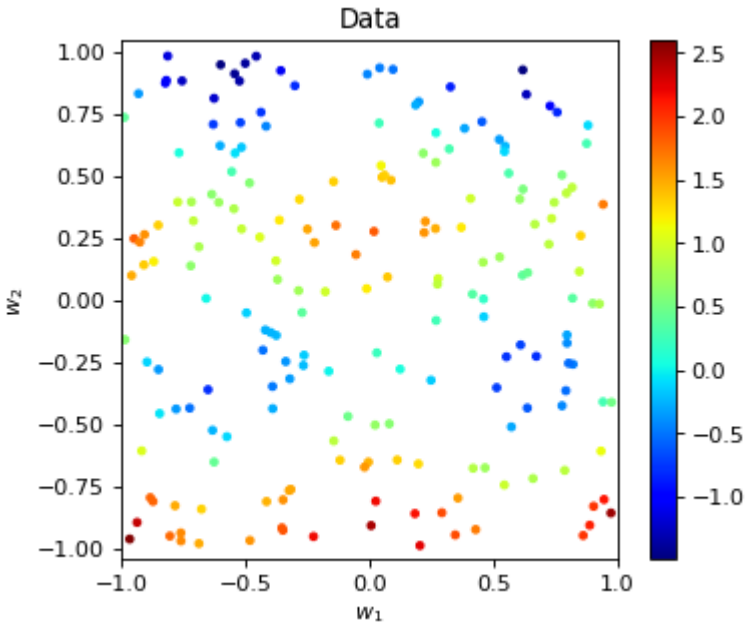
Functions:

- `weighted_knn(w1, w2, k)`

Plots:

- 3 plots of by-hand KNN results
- 3 plots of sklearn.

In [96]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor

# Data generation -- don't change
np.random.seed(42)
N = 200
w1_data = np.random.uniform(-1,1,N)
w2_data = np.random.uniform(-1,1,N)
L_data = np.cos(4*w1_data) + np.sin(5*w2_data) + 2*w1_data**2 - w2_data/2
# (end of data generation)

plt.figure(figsize=(5,4.2),dpi=80)
plt.scatter(w1_data,w2_data,s=10,c=L_data,cmap="jet")
plt.colorbar()
plt.axis("equal")
plt.xlabel("$w_1$")
plt.ylabel("$w_2$")
plt.xlim(-1,1)
plt.ylim(-1,1)
plt.title("Data")
plt.show()
```



Weighted KNN function

Here, define a function, `weighted_knn(w1, w2, k)`, which takes in a point at [`w1` , `w2`] and a `k` value, and returns the weighted KNN prediction.

- As in the lecture activity, data is in the variables `w1_data` , `w2_data` , and `L_data` .
- You can create as many helper functions as you want
- The key difference between unweighted and weighted KNN is summarized below:

Unweighted KNN

1. Find the k data points closest to the target point w
2. Get the output values at each of these points
3. Average these values together: this is the prediction at w

Weighted KNN

1. Find the k data points closest to the target point w
2. Compute the proximity of each of these points as $\text{prox}_i = 1/(\text{distance}(w, w_i) + 1e-9)$
3. For each w_i, multiply prox_i by the output value at w_i, and divide by the sum of all k proximities
4. Add all k of these results together: this is the prediction at w

In [97]:
```
def distance(w1,w2):
    return np.sqrt((w1-w1_data)**2 + (w2-w2_data)**2)

def knn_indices(w1,w2,k):
    d = distance(w1,w2)
    return np.argpartition(d,k)[:k]

def weighted_knn(w1, w2, k):
    dist = distance(w1,w2)
    indices = knn_indices(w1,w2,k)
    val = 0
    prox_sum = 0
    prox = 0
    for j in range(k):
        prox_sum += 1/(dist[indices[j]] + 10**(-9))
    for i in range(k):
        prox = 1/(dist[indices[i]] + 10**(-9))
        w = prox/prox_sum
        val += L_data[indices[i]]*w
    return val
print(weighted_knn(3,2,5))
```

```
-0.6155380702535808
```

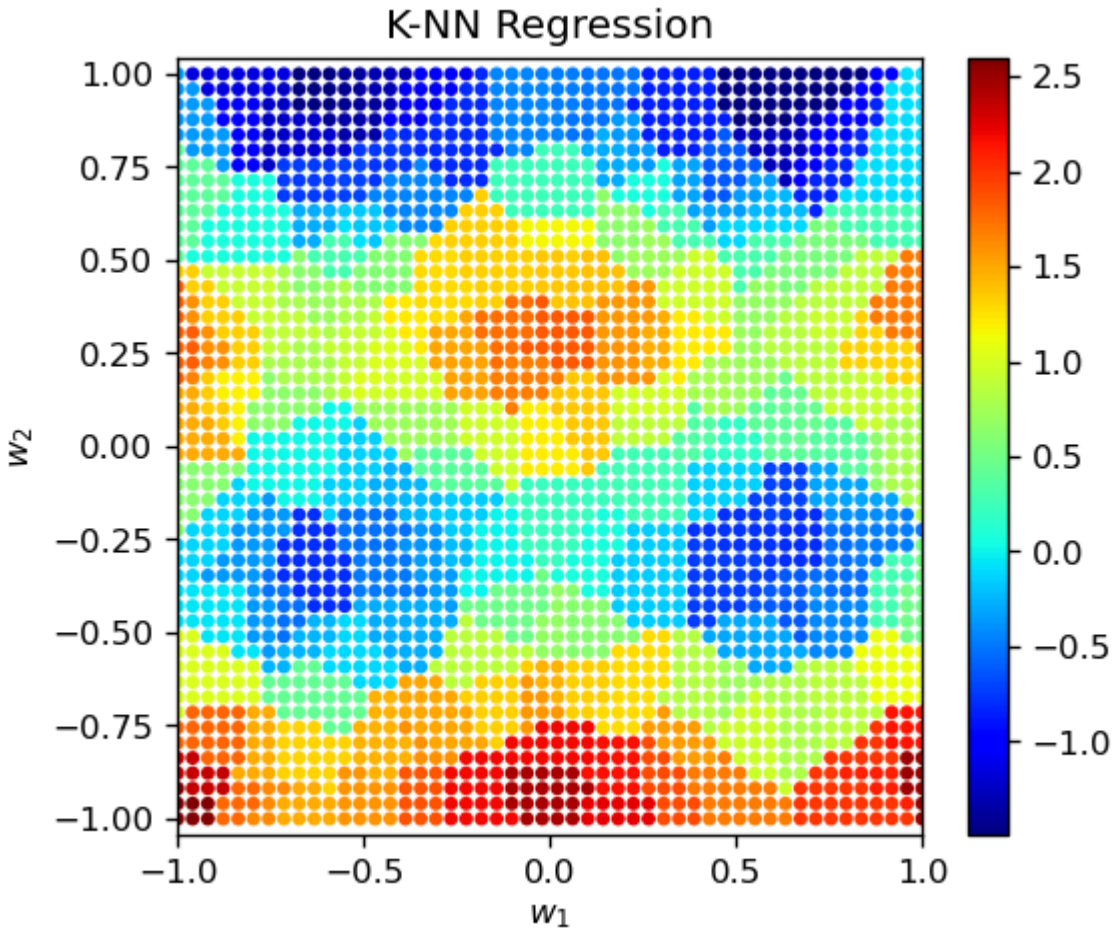
Plotting

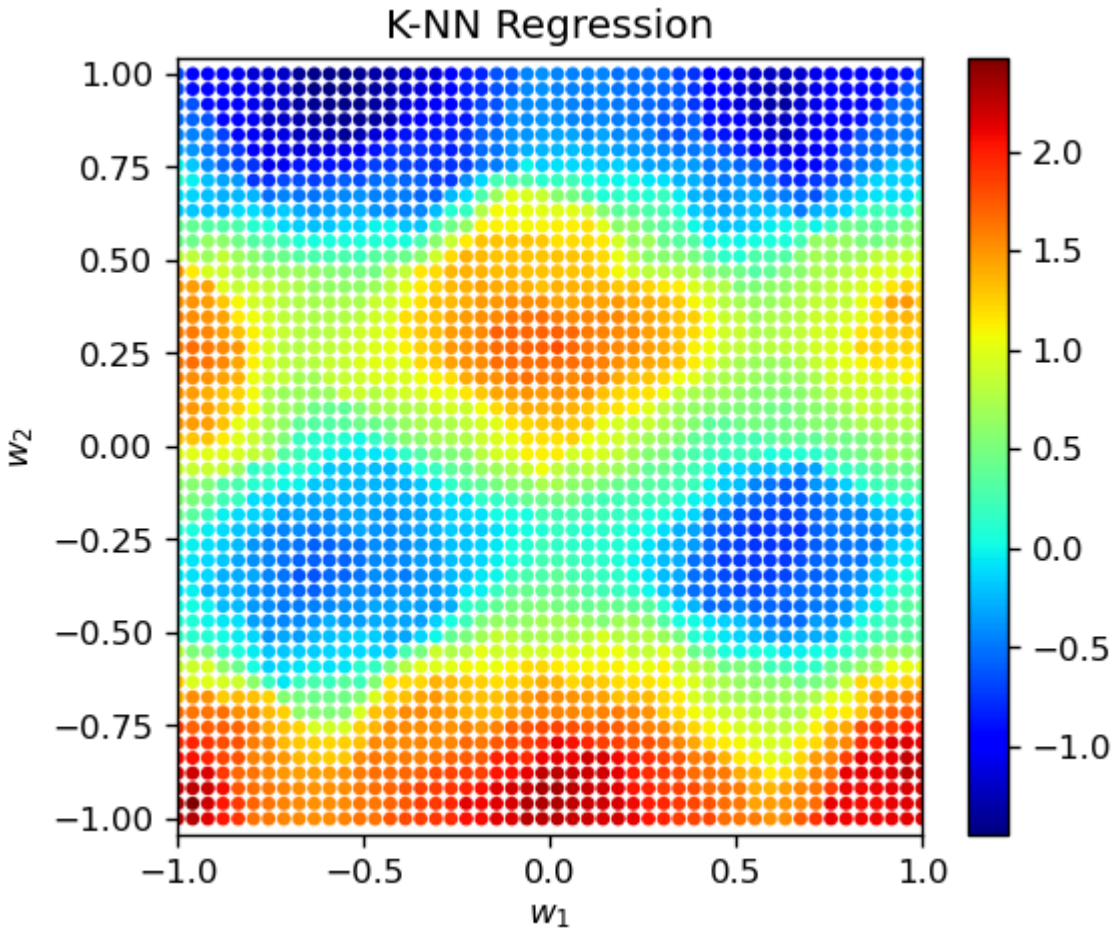
Now create 3 plots showing KNN regressor predictions for k values [1, 5, 25].

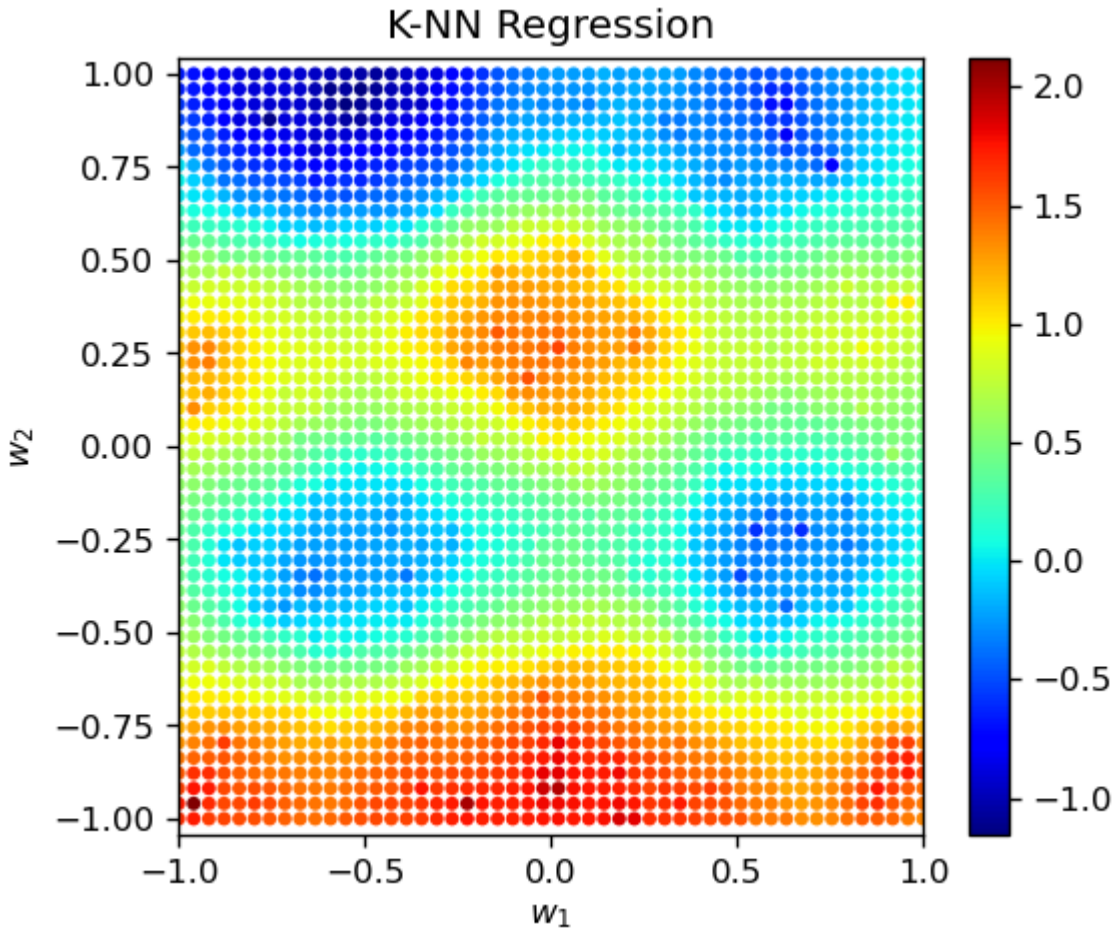
You should plot a 50x50 grid of points on a grid for `w1` and `w2` values between -1 and 1. Consult the lecture activity for how to do this.

We recommend creating a function, e.g. `plot(k)`, so that you need to rewrite less code.

In [117]:
```
w1_vals = np.linspace(-1,1,50)
w2_vals = np.linspace(-1,1,50)
w1s, w2s = np.meshgrid(w1_vals,w2_vals)
w1_grid, w2_grid = w1s.flatten(), w2s.flatten()

k_1 = 1
L_grid_1 = np.zeros_like(w1_grid)

k_2 = 5
L_grid_2 = np.zeros_like(w1_grid)

k_3 = 25
L_grid_3 = np.zeros_like(w1_grid)

for i in range(len(L_grid)):
    L_grid_1[i] = weighted_knn(w1_grid[i], w2_grid[i],k_1)
    L_grid_2[i] = weighted_knn(w1_grid[i], w2_grid[i],k_2)
    L_grid_3[i] = weighted_knn(w1_grid[i], w2_grid[i],k_3)

def plot(L_grid):
    plt.figure(figsize=(5,4.2),dpi=120)
    plt.scatter(w1_grid,w2_grid,s=10,c=L_grid,cmap="jet")
    plt.colorbar()
    plt.axis("equal")
    plt.xlabel("$w_1$")
    plt.ylabel("$w_2$")
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.title("K-NN Regression")
    plt.show()


plot(L_grid_1)
plot(L_grid_2)
plot(L_grid_3)


# Visualize results for k = 1, 5, and 25
```


K-NN Regression
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.0
−0.5
0.0
0.5
1.0
w_1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0


K-NN Regression
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.0
−0.5
0.0
0.5
1.0
w_1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0



Using SciKit-Learn

We can also use sklearn's `KNeighborsRegressor()` , which is a very efficient implementation of KNN regression.

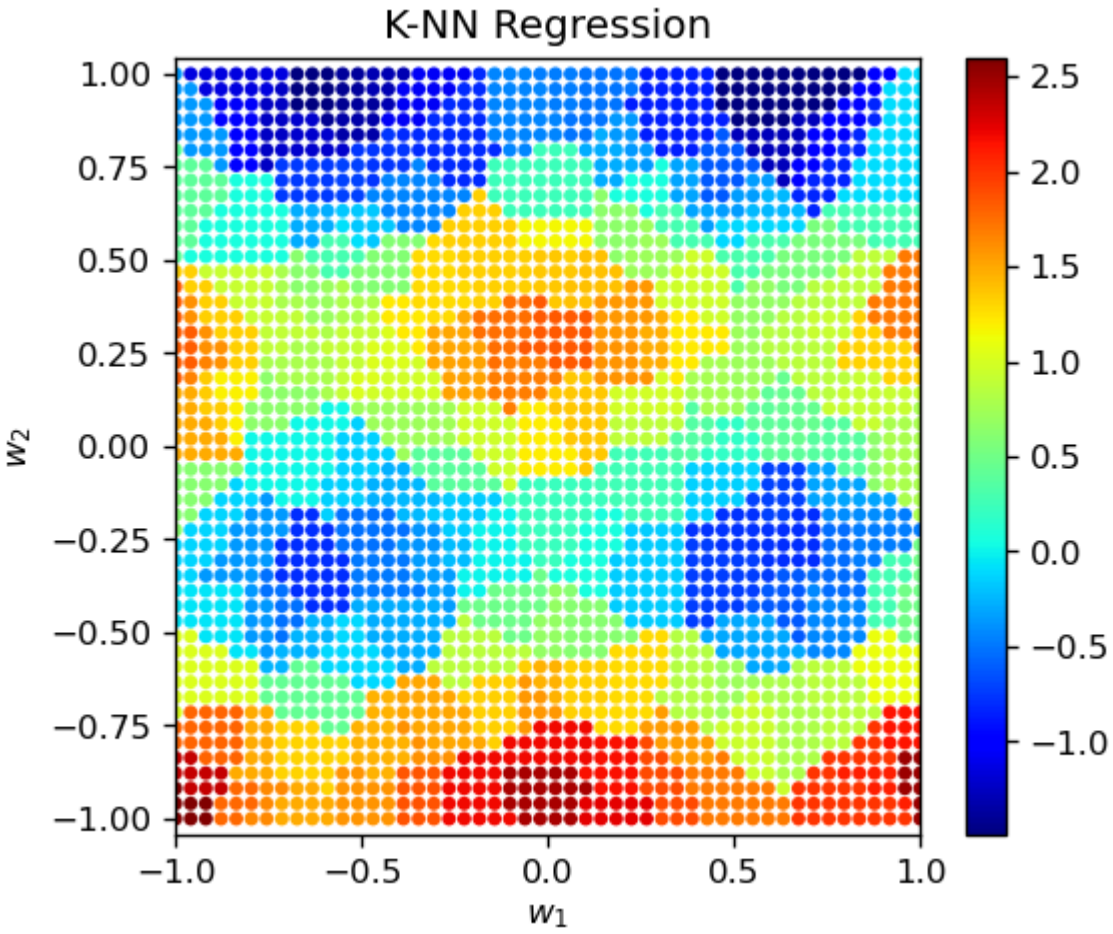
The code to do this has been done for one case below. First, make note of how this is done.

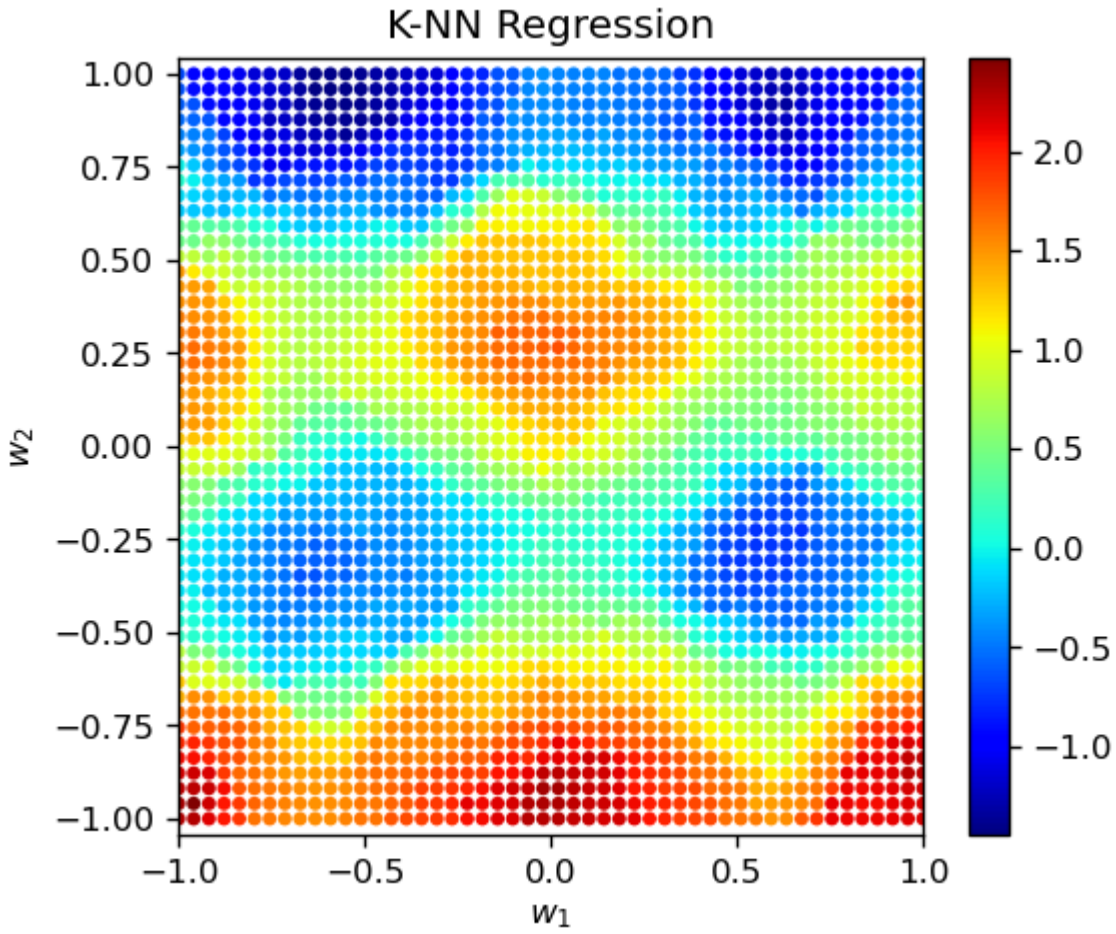
In [118]:
```
model = KNeighborsRegressor(n_neighbors = 1, weights="distance")
X = np.vstack([w1_data,w2_data]).T
model.fit(X, L_data)

# Get a prediction at a point (0, 0):
print(model.predict(np.array([[0,0]])))
```

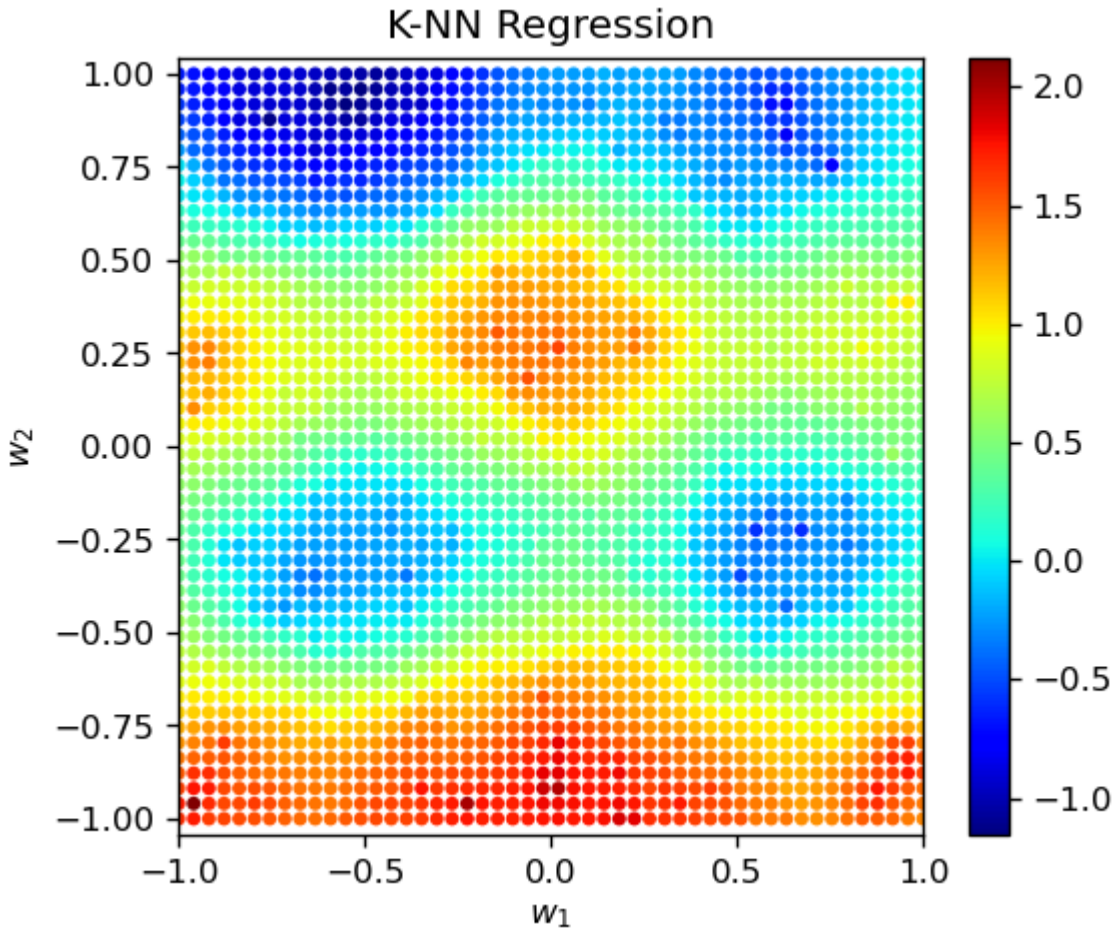
```
[1.19743607]
```

Now create 3 plots for the same values of k as before, using this KNN implementation instead. You can make sure these are visually the same as your from-scratch KNN regressor.

In [120]:
```
def plot(k):
    model = KNeighborsRegressor(n_neighbors = k, weights="distance")
    X = np.vstack([w1_data,w2_data]).T
    model.fit(X,L_data)
    w1_vals = np.linspace(-1,1,50)
    w2_vals = np.linspace(-1,1,50)
    w1s, w2s = np.meshgrid(w1_vals,w2_vals)
    w1_grid, w2_grid = w1s.flatten(), w2s.flatten()
    L_grid = np.zeros_like(w1_grid)

    for i in range(len(L_grid)):
        L_grid[i] = model.predict([[w1_grid[i],w2_grid[i]]])
    plt.figure(figsize=(5,4.2),dpi=120)
    plt.scatter(w1_grid,w2_grid,s=10,c=L_grid,cmap="jet")
    plt.colorbar()
    plt.axis("equal")
    plt.xlabel("$w_1$")
    plt.ylabel("$w_2$")
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.title("K-NN Regression")
    plt.show()



plot(1)
plot(5)
plot(25)

# Visualize sklearn results for k = 1, 5, and 25
```


K-NN Regression
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.0
−0.5
0.0
0.5
1.0
w_1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0


K-NN Regression
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.0
−0.5
0.0
0.5
1.0
w_1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0


K-NN Regression
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
w_2
−1.0
−0.5
0.0
0.5
1.0
w_1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0

In [ ]: